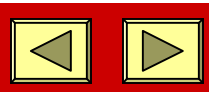
Towards a Generic Crypto API
Presentation to NIST

November 8, 2001

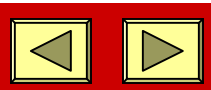
SUMMARY

AGENDA



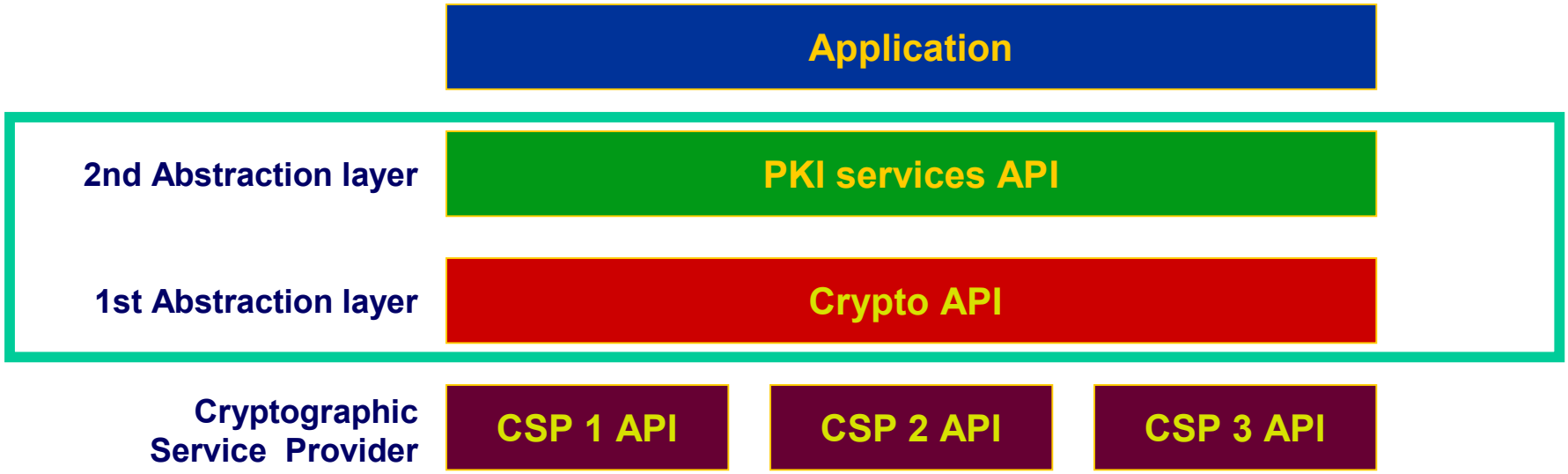
Summary

- A multiplicity of crypto APIs is an obstacle to the widespread adoption of cryptography by applications to protect data
- Ideally, a generic crypto API would provide an abstraction layer that would “unify” the interface to the cryptographic service providers (CSPs) and be sufficiently high level that non-cryptographer application developers could use it
- We present an approach that has two levels of abstraction between the CSP and the application to achieve these objectives

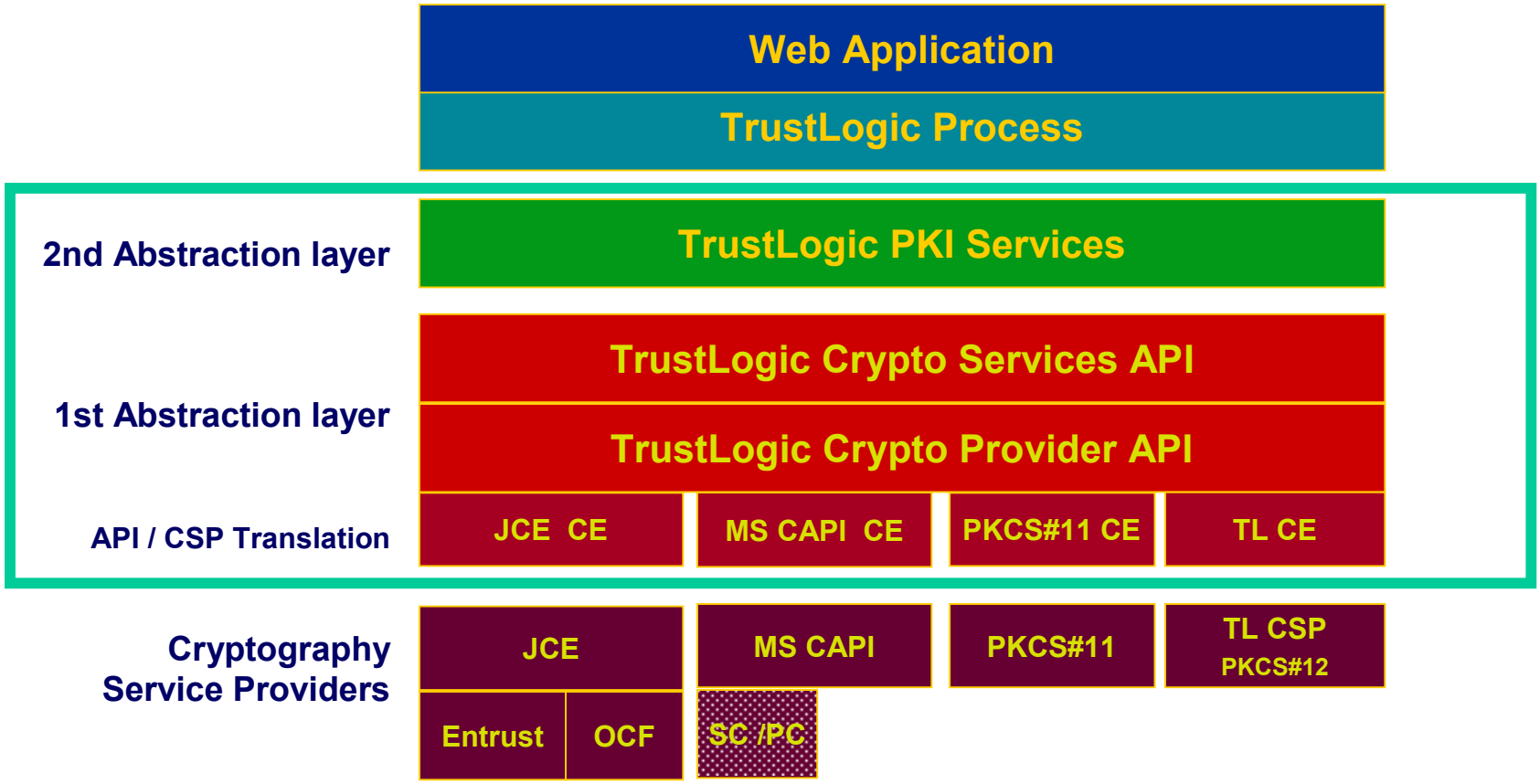


The architecture is straightforward.

- 1st abstraction layer is the generic crypto API that unifies the CSPs from the developer's perspective.
- 2nd abstraction layer is PKI services that make PKI easily accessible to the application.

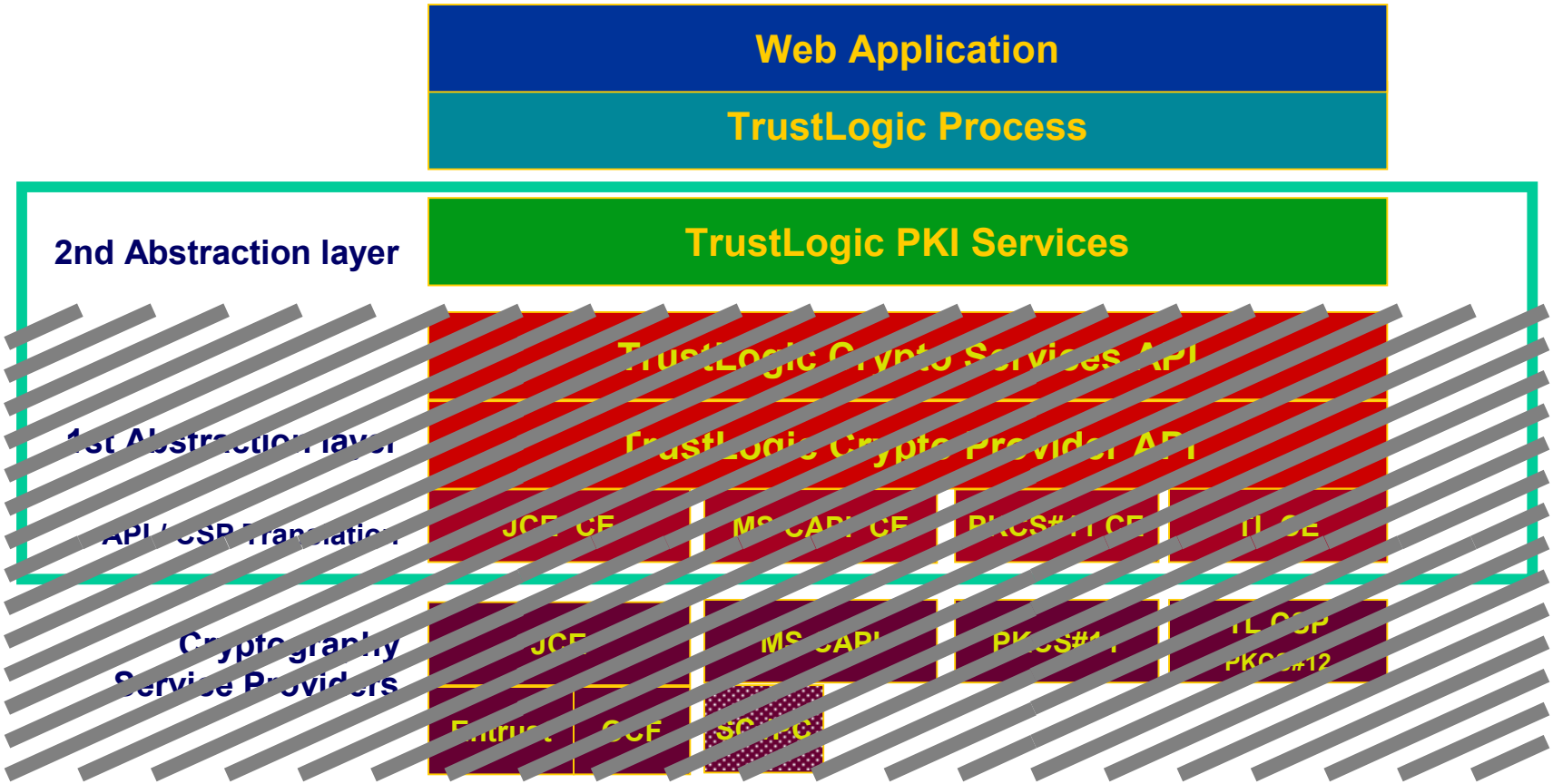


In our implementation of this architecture we have broken this down into additional layers to simplify the developer's work and add flexibility.

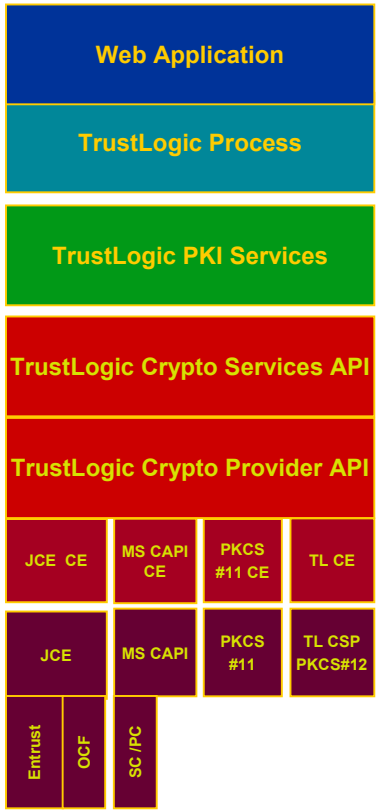


Let's build this up.

The good news is all the crypto is opaque to the application developer.



The Crypto Provider API is the unifying layer that eliminates the need for the application developer to manage the multiple APIs of the CSP providers.

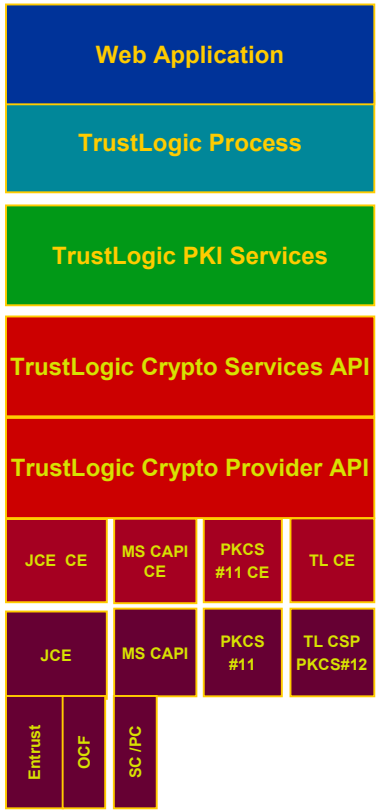


- CDSA like architecture
- Create crypto context
- Create context for crypto operation
- Perform operation
- Support for single pass and staged cryptography

The generic API covers basic crypto operations.

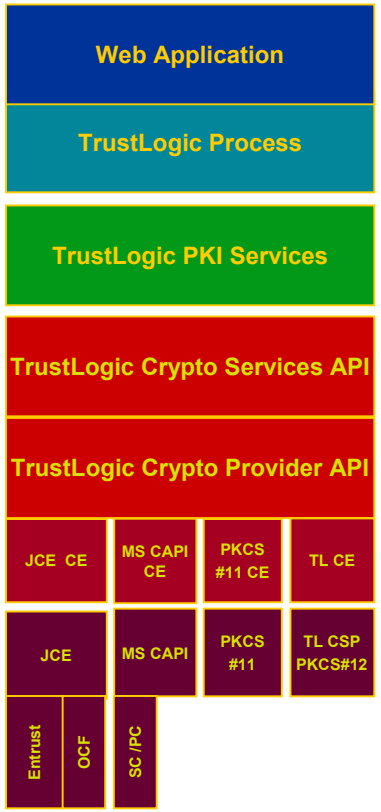
Algs and Context	Single Pass	Staged Operations	
createCSPContext	deriveKey	hashDataInit	signDataInit
	generateKey	hashDataUpdate	signDataUpdate
getSupportedAlgs	generateSharedKey	hashDataFinal	signDataFinal
isAlgSupported	generateKeyPair		
generateAlgorithmParams	generateKeyPair	encryptDataInit	generateMacInit
generateRandom	wrapKey	encryptDataUpdate	generateMacUpdate
	unwrapKey	encryptDataFinal	generateMacFinal
createCryptoContextDigest	encrypt		
createCryptoContextSignature	decrypt	decryptDataInit	verifyMacInit
createCryptoContextSymmetric	digest	decryptDataUpdate	verifyMacUpdate
createCryptoContextRandomGen	sign	decryptDataFinal	verifyMacFinal
createCryptoContextAsymmetric	verify		
createCryptoContextDeriveKey	generateMac	verifyDataFinal	
createCryptoContextKeyGen	verifyMac	verifyDataUpdate	
createCryptoContextMac		verifyDataInit	
createCryptoContextDH			

The “glue” between this “generic” API and the CSPs are individual “translation” modules that interpret the API to the specific CSP.



- CSP specific
- Re-interpret generic call in CSP specific call, using parameters from generic call and information from the specific operation context
- Where necessary “bridge” software technologies, e.g. Java to *.dll for MS CAPI and software to hardware, e.g. on card crypto engines (software/ hardware mix)

To facilitate development, we have created a higher level layer that aggregates instructions at the lower Crypto Provider CSP level.



- High level calls that amalgamate multiple instructions at the Crypto Provider level
- “init” calls that create a crypto “helper”, a wrapper around the context such that the developer need only identify the crypto operation to be performed

Crypto Services provide a single call for Key operations that otherwise would need a large number of instructions, and create “helpers” that set up the context for a crypto operation.

Create Crypto “Helpers”
initDigest
initEncryptWithCert
initEncryptWithKey
initEncryptWithKey
initEncryptWithPhrase
initEncryptWithWrappedKey
initMac
initSign
initVerify
initVerifyWithCert

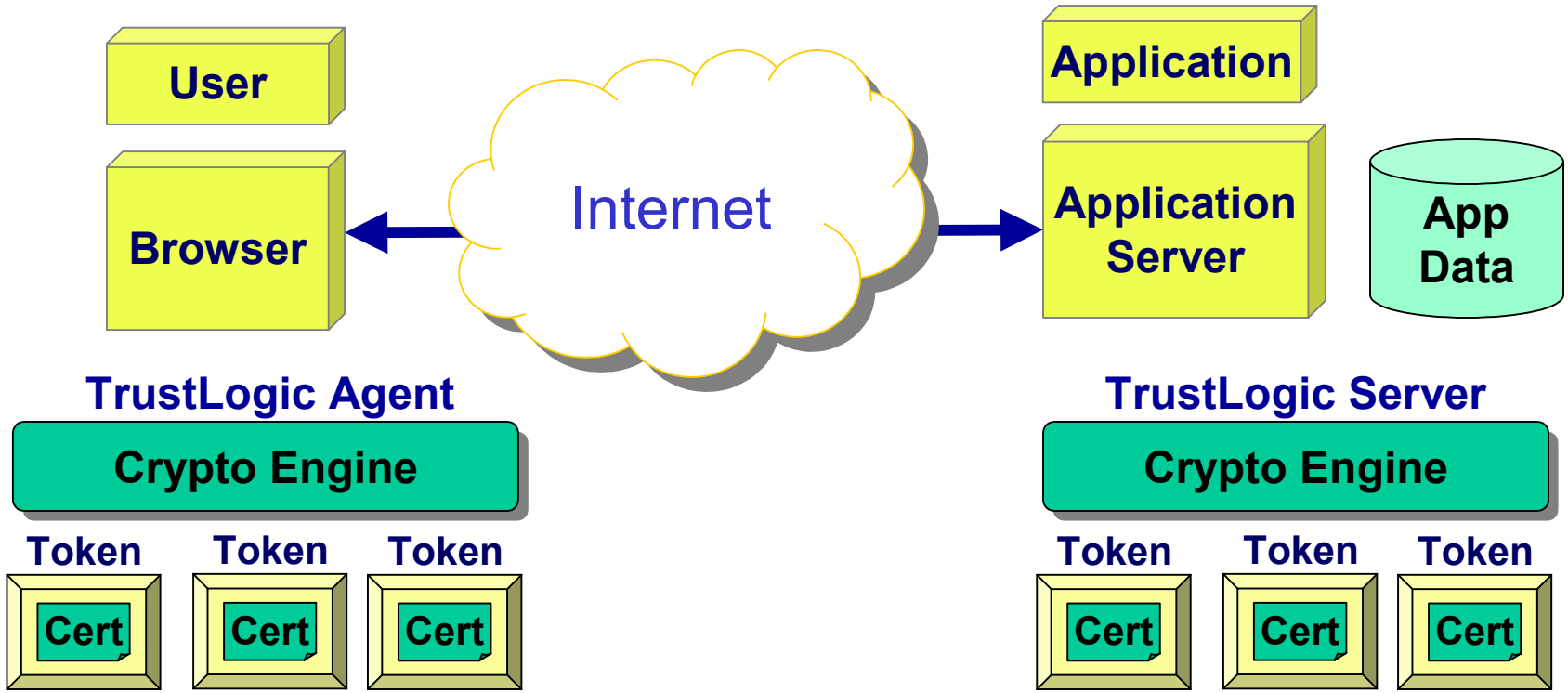
Key Operations
addTokenProvider
changeKeyWrap
deriveKey
generateKey
generateKey
generateKeyPair
loadCSPFromAlgID
loadCSPFromClass
passwordUnwrapPrivateKey
passwordWrapPrivateKey
unwrapKey
wrapKey
wrapKeyWithPublicKey

The Crypto Services and Crypto Provider layers provide a simplified crypto interface,

BUT

- How does a web application get data signed and encrypted, server-side and at the browser?
  - Where is the functionality to accomplish this
  - How does a web application communicate with the crypto?
  - Where are the services to handle user authentication, including certificate checking?
- And, how does the Crypto Provider know which CSP to use?

Lets step back from our crypto engine to see where this fits in to the bigger picture. Two parties (app & user) are trying to use PKI. Both may have multiple tokens – with related CSPs.



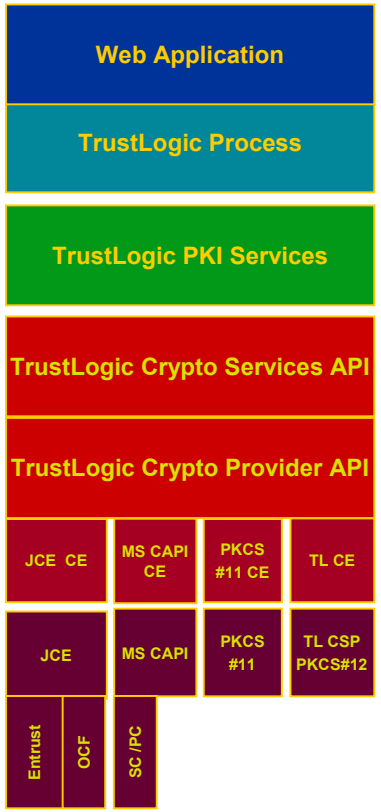
To make it all work we add 3 pieces of XML. The KeyIndex, the XMLAction, and the TrustLogic Context.

Key Index	XML Action	TL Context
• XML associated to an identity • Stored "locally" under encrypted password • Holds association between keys and applications • Holds token information about keys • KeyInfo complies with XML DSIG	• Crypto instructions in XML for specific data exchange • Describes key, algorithm to use when signing and /or encrypting data • Stored in secure data repository • Schema to be offered to IETF /W3C as RFC	• Session dependent XML that holds data about the current user • Role, Certificates, Token, etc. • Signed by both parties

With these 3 pieces of XML, we can start to tie things together.

- The application wants to send signed and encrypted data to the user:
- It retrieves the XMLAction that holds the cryptographic instructions
- From the XMLAction it determines which keys are to be used
- It looks up the key in its key index to find out what token is being used
- It can now call Crypto Services to set up a crypto context with the name of the token
- The token indicates which CSP the Crypto Provider layer will use

We make this a lot simpler than that, to achieve the foregoing, the web application developer uses TrustLogic services.



- TL Services: a high level API for PKI services to web applications
- Provide services for authentication, data protection, evidentiary proof, TL context management, and various utilities

For the example of sending signed and encrypted data to the browser the call is simply protectXML (the data is described in XML).

protectXML (protectID, requestID, tlContext, xmlIn)

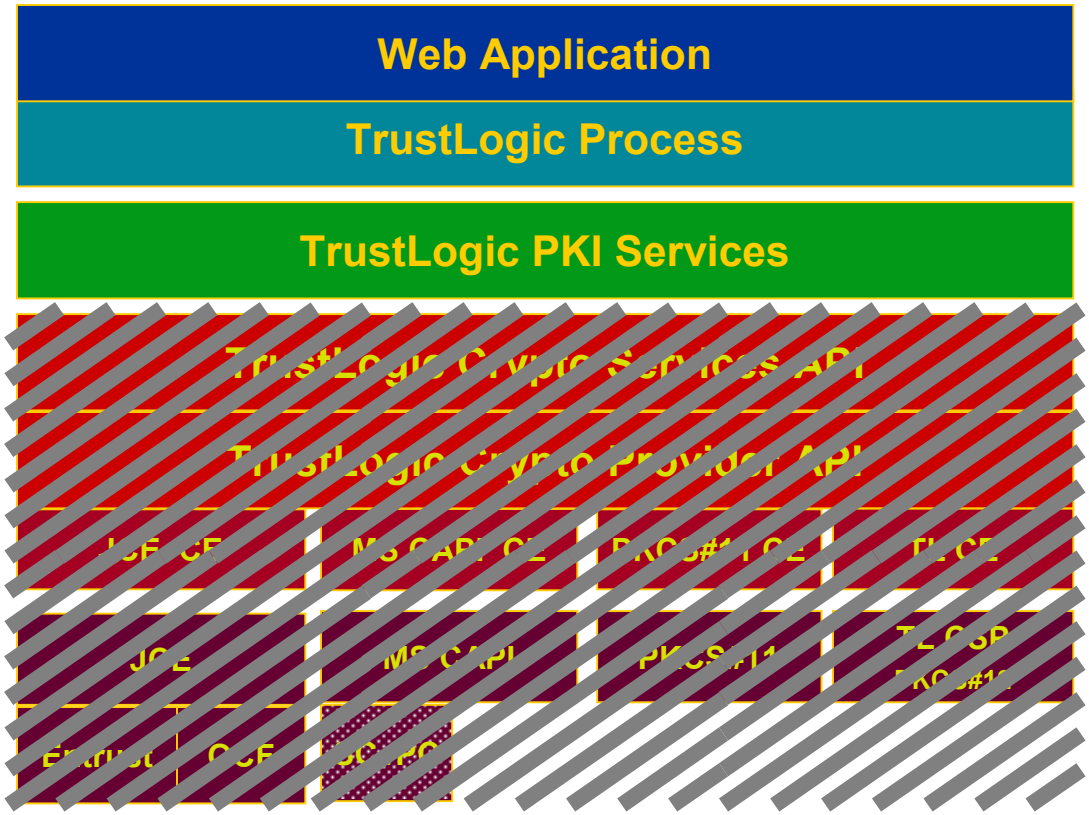
Where

- **protectID: is the name of the XMLAction describing the cryptography to be performed server-side,**
- **requestID: is the name of the XMLAxction to be embedded in the form, describing the cryptography to be applied by TL Agent**
- **XMLin: is the data, described in XML**

Example:

```
tlServices.protectXML("ServerProtectConfirmForm",
"BrowserUpdateConfirmForm", tlContext, xmlIn);
```

So, while we have a generic crypto API that provides a uniform interaction with multiple CSPs, we do not expose it in our product. The visible level is TrustLogic services.



TrustLogic services provide a range of functionality needed by web applications. They are a generic API for delivering PKI services to web applications.

Authentication

createChallenge
verifyChallenge
checkCATrustLevel
validateCertificate
setLogonRole

Data Management

protectXML
unprotectXML
verifyXML
verifyActions
importEncryptedtXML
exportEncryptedXML

Context Management

getSigningCertificate
getTLCertificate
getRuleData
addAttributeToContext
addAttributesToContext

User Management

create Registration
checkRegistration
writeRegistrationData

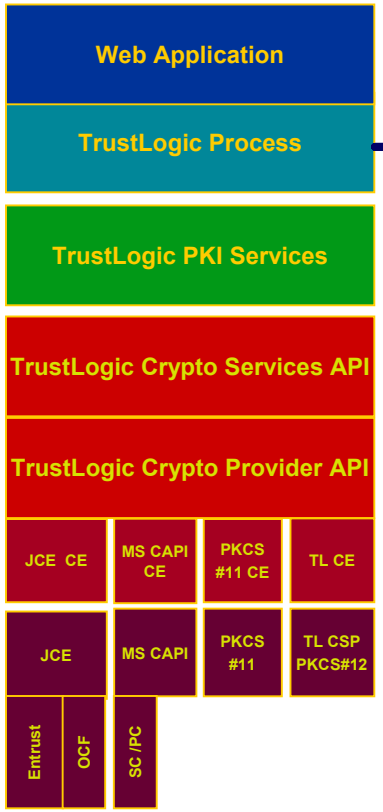
Utilities

attachProperties
readData
setUserRole
SendEMail

Audit

writeAuditEvent

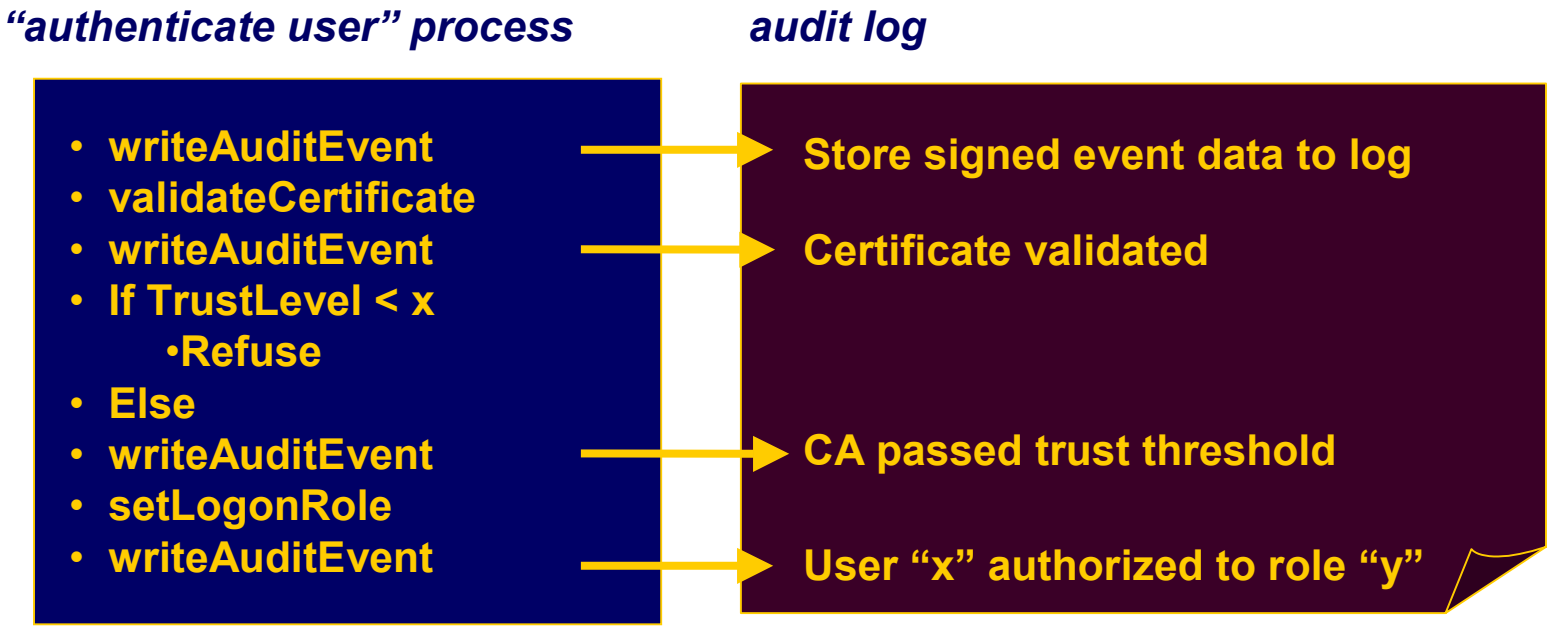
The final layer of our cake is the TrustLogic process. It has nothing to do with crypto, but its worth a mention to complete the picture.



- The TrustLogic process pulls together a series of TrustLogic Services and rules
- Processes can be generic across all applications, such as authenticate user, or specific to an application, such as confirm order

The "process" is in fact the application interface. A process is built from calling TrustLogic services, and invoking rules. Data is "logged" according the application requirement for proof.

ILLUSTRATIVE PROCESS



Thank you.

Detailed APIs are available on request:

Skip Chapman
schapman @ conclusive.com
703 734 3000 ext 101

Matthew McKennirey
mmckennirey @ conclusive.com
703 734 3000 ext 120